山东大学 计算机科学与技术 学院

操作系统 课程实验报告

学号:201900130059	姓名: 孙奇	班级: 2019 级 1 班
实验题目: 线程和管道通信		
实验学时: 2	实验日期: 2021/10/22	

实验目的:

通过 Linux 系统中线程和管道通信机制的实验，熟悉 pthread 线程库的使用，加深对于线程控制和管道通信概念的理解，观察和体验并发线程间的通信和协作的效果，练习基于 pthread 线程库、利用无名管道进行线程通信的编程和调试技术。

硬件环境:

CPU: Intel i5-9300H

GPU: UHD630

软件环境:

Ubuntu 18.04

实验步骤与内容:

1. 实验说明:

（1） 线程说明: 线程是在共享内存中并发执行的多道执行路径，它们共享一个进程的资源，如进程程序段、文件描述符和信号量，但有各自的执行路径和堆栈。线程的创建无需像进程那样重新申请系统资源，线程在上下文切换时也无需像进程那样更换内存映像。多线程的并发执行即避免了多进程并发的上下文切换的开销又可以提高并发处理的效率。

（2） 管道通信机制: 管道 pipe 是实现进程间通信的一种机制。在内存中临时建立的管道称为无名管道，在磁盘上建立的管道实质上是一个特殊类型的文件，称为有名管道。无名管道随着进程的撤消而消失，有名管道则可以长久保存，shell 命令符“|”建立的就是无名管道，而 shell 命令 mkfifo 建立的是有名管道。如果一个进程通过管道的一端向管道写入数据，另一进程可以在管道的另一端从管道中读取数据，则实现了 两个进程之间的通信。管道以半双工方式工作，即它的数据流是单方向的。因此使用一个管道一般的规则是，读管道数据的进程保留读出端，关闭管道写入端；而写管道进程保留写入端，关闭其读出端。管道既可以采用同步方式工作也可以采用异步方式工作，其中管道默认采用同步工作方式，采用了生产者/消费者模型的同步工作机制。

2. 独立实验:

设有二元函数 f(x,y) = f(x) + f(y) 其中:

- f(x)=1 (x=1)
- f(x) = f(x-1) * x (x >1)
- f(y)=1 (y=1,2)
- f(y) = f(y-1) + f(y-2) (y> 2)

请基于无名管道，利用 pthread 线程库编程建立 3 个并发协作线程，它们分别 完成 f(x,y)、f(x)、f(y)

3. 实现思路：
建立三个线程 f(x)，f(y)，f(x, y)，分别完成三个函数；建立两个管道，作为 f(x, y) --- f(x)的读写管道和 f(x, y) --- f(y)的读写管道，主线程读取输入 x, y，然后 pthread_create()建立三个子线程，建立完成后 pthread_join()阻塞主线程运行，在完成子线程运行后，回收子线程资源，然后继续执行主线程，结束程序；

4. 执行过程：

```
x = 4
y = 5
==================Thread f(x)====================
f(x) write 24 to pipe_x[0]
==================Thread f(y)====================
f(y) write 5 to pipe_y[0]
==================Thread f(x, y)==================
f(x, y) read 24 from pipe_x[1]
==================Thread f(x, y)==================
f(x, y) read 5 from pipe_y[1]
f(x, y) = 29
```

结论分析与体会：
1. 线程协作和通信的特征和功能：
（1） 线程协作并发进行，将大任务进行拆分，各线程完成单独的子任务，子任务完成了那么全部的线程合作完成了大任务；
（2） 线程通信通过管道实现，管道作为消费者生产者模型，保证了线程执行的同步和等待，体现了线程子任务执行之间的逻辑顺序，保证了大任务的顺利完成；

2. 管道机制的机理：
管道实际上是一种固定大小的缓冲区，管道对于管道两端的进程而言，就是一个文件；它类似于通信中半双工信道的进程通信机制，一个管道可以实现双向的数据传输，而同一个时刻只能最多有一个方向的传输，不能两个方向同时进行。管道是一个生产者-消费者模型，当管道空时，只能 write 入管道，读阻塞，当管道满时，只能 read 出管道，写阻塞；

3. 管道对进/线程的辅助：
进/线程之间的信息传递通过管道实现，具体上，进/线程将数据写入管道，另外的进/线程从管道读出相应数据，这就实现了数据在多个线程之间的传递；

4. 如果创建线程后，不调用 pthread_join()会发生什么现象?
答：如果不调用 pthread_join()，程序得不到正确输出，主线程不会等待子线程执行结束，会继续执行，父子线程并发执行，父线程结束后，子线程可能未执行完，子线程的资源得不到回收成为了僵尸线程；

附录：程序源代码

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

// write: pipe[1]
// read: pipe[0]
int pipe_x[2];
int pipe_y[2];

typedef struct {
	int x;
	int y;
} pair;

static void fx(int *x) {
	int i;
	int res = 1;
	for (i = 1; i <= *x; i++)
		res *= i;
	write(pipe_x[1], &res, sizeof(int));
	printf("==================Thread f(x)==================\n");
	printf("f(x) write %d to pipe_x[0]\n", res);
	close(pipe_x[1]);
}

static int solve(int y) {
	if (y == 1 || y == 2)
		return 1;
	return solve(y - 1) + solve(y - 2);
}

static void fy(int *y) {
	int res = solve(*y);
	write(pipe_y[1], &res, sizeof(int));
	printf("==================Thread f(y)==================\n");
	printf("f(y) write %d to pipe_y[0]\n", res);
	close(pipe_y[1]);
}

static void fxy(int *x, int *y) {
	int fx = 0, fy = 0, res = 0;
	read(pipe_x[0], &fx, sizeof(int));
	printf("==================Thread f(x, y)==================\n");
	printf("f(x, y) read %d from pipe_x[1]\n", fx);
	read(pipe_y[0], &fy, sizeof(int));
```

```
	printf("==================Thread f(x, y)==================\n");
	printf("f(x, y) read %d from pipe_y[1]\n", fy);
	res = fx + fy;
	printf("f(x, y) = %d\n", res);
	close(pipe_x[0]);
	close(pipe_y[0]);
}

int main() {
	int x = 0, y = 0, status = 0;
	pair xy;
	pthread_t fx_tid, fy_tid, fxy_tid;

	if (pipe(pipe_x) < 0) {
		perror("Pipe_x Create Failed.\n");
		exit(EXIT_FAILURE);
	}
	if (pipe(pipe_y) < 0) {
		perror("Pipe_y Create Failed.\n");
		exit(EXIT_FAILURE);
	}

	printf("x = ");
	scanf("%d", &x);
	printf("y = ");
	scanf("%d", &y);
	xy.x = x;
	xy.y = y;
	status = pthread_create(&fx_tid, NULL, (void *)fx, (void *) &x);
	if (status) {
		perror("Thread f(x) Create Failed.\n");
		exit(EXIT_FAILURE);
	}
	status = pthread_create(&fy_tid, NULL, (void *)fy, (void *) &y);
	if (status) {
		perror("Thread f(y) Create Failed.\n");
		exit(EXIT_FAILURE);
	}
	status = pthread_create(&fxy_tid, NULL, (void *)fxy, (void *) &xy);
	if (status) {
		perror("Thread f(x, y) Create Failed.\n");
		exit(EXIT_FAILURE);
	}

	pthread_join(fx_tid, NULL);
```

```
    pthread_join(fy_tid, NULL);
    pthread_join(fxy_tid, NULL);

    exit(EXIT_SUCCESS);
}
```